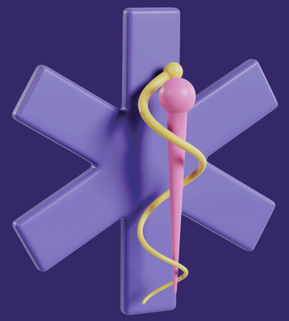
IE0005 Mini Project

# CARDIOVASCULAR



Problem Statement

# PREDICTING THE RISK OF CARDIOVASCULAR DISEASES THROUGH LIFESTYLE HABITS

# WHAT WE WILL COVER

1. Data Cleaning
2. Exploratory Data Analysis & Visualisation
3. Machine Learning Models
4. Conclusion

# ORGANISING DATA

## Initial dataset

| | id;age;gender;height;weight;ap_hi;ap_lo;cholesterol;gluc;smoke;alco;active;cardio |
|---|---|
| **0** | 0;18393;2;168;62.0;110;80;1;1;0;0;1;0 |
| **1** | 1;20228;1;156;85.0;140;90;3;1;0;0;1;1 |
| **2** | 2;18857;1;165;64.0;130;70;3;1;0;0;0;1 |
| **3** | 3;17623;2;169;82.0;150;100;1;1;0;0;1;1 |
| **4** | 4;17474;1;156;56.0;100;60;1;1;0;0;0;0 |

# ORGANISING DATA

**Unnecessary variable**

| | id | age | gender | height | weight | ap_hi | ap_lo | cholesterol | gluc | smoke | alco | active | cardio |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 18393 | 2 | 168 | 62.0 | 110 | 80 | 1 | 1 | 0 | 0 | 1 | 0 |
| **1** | 1 | 20228 | 1 | 156 | 85.0 | 140 | 90 | 3 | 1 | 0 | 0 | 1 | 1 |
| **2** | 2 | 18857 | 1 | 165 | 64.0 | 130 | 70 | 3 | 1 | 0 | 0 | 0 | 1 |
| **3** | 3 | 17623 | 2 | 169 | 82.0 | 150 | 100 | 1 | 1 | 0 | 0 | 1 | 1 |
| **4** | 4 | 17474 | 1 | 156 | 56.0 | 100 | 60 | 1 | 1 | 0 | 0 | 0 | 0 |

# ORGANISING DATA

| | age | gender | height | weight | ap_hi | ap_lo | cholesterol | gluc | smoke | alco | active | cardio | BMI |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 50 | 2 | 168 | 62.0 | 110 | 80 | 1 | 1 | 0 | 0 | 1 | 0 | 21.967 |
| **1** | 55 | 1 | 156 | 85.0 | 140 | 90 | 3 | 1 | 0 | 0 | 1 | 1 | 34.928 |
| **2** | 52 | 1 | 165 | 64.0 | 130 | 70 | 3 | 1 | 0 | 0 | 0 | 1 | 23.508 |
| **3** | 48 | 2 | 169 | 82.0 | 150 | 100 | 1 | 1 | 0 | 0 | 1 | 1 | 28.710 |
| **4** | 48 | 1 | 156 | 56.0 | 100 | 60 | 1 | 1 | 0 | 0 | 0 | 0 | 23.011 |

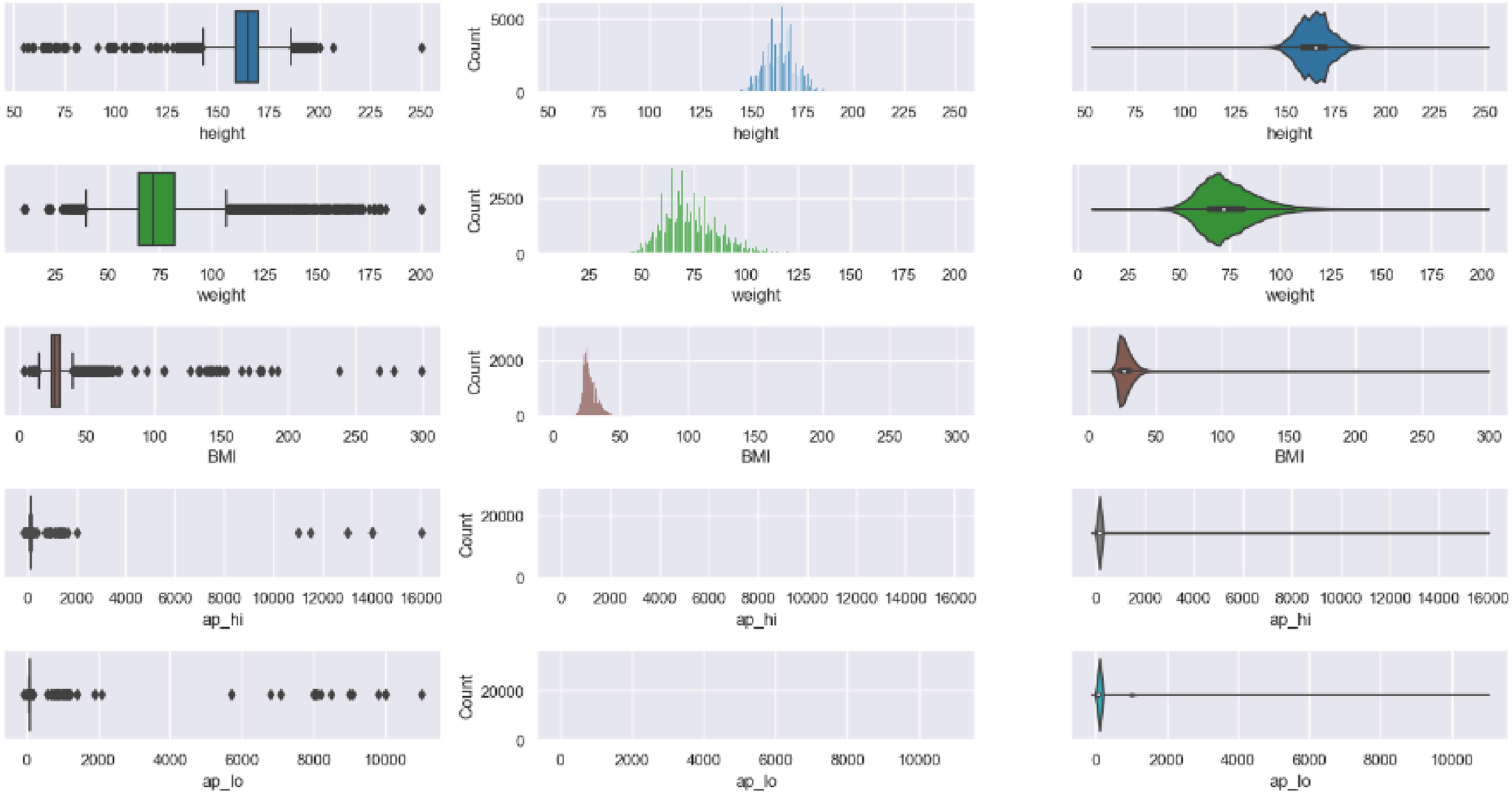
**Changed from days to years**

$$\text{BMI} = \frac{\text{Weight in kilogram}}{\left(\text{Height in meter}\right)^2}$$

# CLEARING ANOMALIES

| | age | gender | height | weight | ap_hi | ap_lo |
|---|---|---|---|---|---|---|
| **count** | 70000.000000 | 70000.000000 | 70000.000000 | 70000.000000 | 70000.000000 | 70000.000000 |
| **mean** | 53.338686 | 1.349571 | 164.359229 | 74.205690 | 128.817286 | 96.630414 |
| **std** | 6.765294 | 0.476838 | 8.210126 | 14.395757 | 154.011419 | 188.472530 |
| **min** | 30.000000 | 1.000000 | 55.000000 | 10.000000 | -150.000000 | -70.000000 |
| **25%** | 48.000000 | 1.000000 | 159.000000 | 65.000000 | 120.000000 | 80.000000 |
| **50%** | 54.000000 | 1.000000 | 165.000000 | 72.000000 | 120.000000 | 80.000000 |
| **75%** | 58.000000 | 2.000000 | 170.000000 | 82.000000 | 140.000000 | 90.000000 |
| **max** | 65.000000 | 2.000000 | 250.000000 | 200.000000 | 16020.000000 | 11000.000000 |

# CLEARING ANOMALIES



# CLEARING ANOMALIES

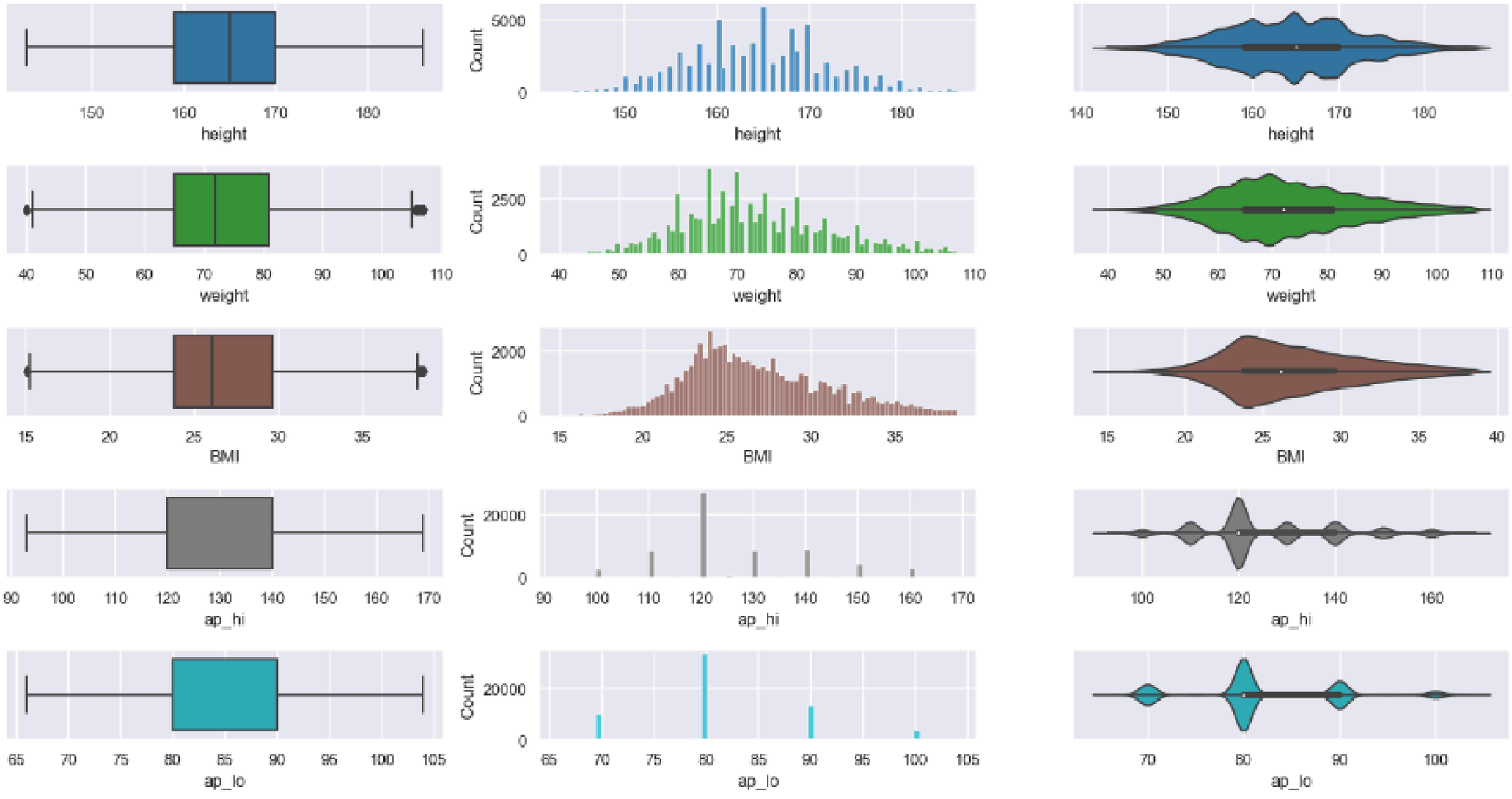
```python
# Function to flag outliers using IQR method
def flag_outliers(df, var, degree=1.5):
    # Calculate the interquartile range (IQR)
    lq = df[var].quantile(0.25)
    uq = df[var].quantile(0.75)
    iqr = uq - lq

    # Calculate the lower and upper bounds
    lowerbound = lq - degree * iqr
    upperbound = uq + degree * iqr

    return lowerbound, upperbound

# Function to drop outliers and plot the boxplot, histogram, and violin plot
def drop_outliers_and_plot(df, var, lowerbound, upperbound, ax, color):
    # Filter out the rows with outliers and create a cleaned DataFrame
    df_cleaned = df[(df[var] > lowerbound) & (df[var] < upperbound)]
```

# CLEARING ANOMALIES



# CATEGORISING DATA

| | age | gender | height | weight | ap_hi | ap_lo | cholesterol | gluc | smoke | alco | active | cardio | BMI | Age Group | BMI Group | MAP | MAP Group |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 50 | 2 | 168 | 62.0 | 110 | 80 | 1 | 1 | 0 | 0 | 1 | 0 | 21.967 | 50-54 | Normal Weight | 90.000000 | Optimal |
| **1** | 55 | 1 | 156 | 85.0 | 140 | 90 | 3 | 1 | 0 | 0 | 1 | 1 | 34.928 | 55-59 | Obese Class I | 106.666667 | Grade 1 Hypertension |
| **2** | 52 | 1 | 165 | 64.0 | 130 | 70 | 3 | 1 | 0 | 0 | 0 | 1 | 23.508 | 50-54 | Normal Weight | 90.000000 | Optimal |
| **3** | 48 | 2 | 169 | 82.0 | 150 | 100 | 1 | 1 | 0 | 0 | 1 | 1 | 28.710 | 45-49 | Overweight | 116.666667 | Grade 1 Hypertension |
| **4** | 48 | 1 | 156 | 56.0 | 100 | 60 | 1 | 1 | 0 | 0 | 0 | 0 | 23.011 | 45-49 | Normal Weight | 73.333333 | Optimal |

# CATEGORISING DATA

| | age | gender | height | weight | ap_hi | ap_lo | cholesterol | gluc | smoke | alco | active | cardio | BMI | Age Group | BMI Group | MAP | MAP Group |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 50 | 2 | 168 | 62.0 | 110 | 80 | 1 | 1 | 0 | 0 | 1 | 0 | 21.967 | 50-54 | Normal Weight | 90.000000 | Optimal |
| **1** | 55 | 1 | 156 | 85.0 | 140 | 90 | 3 | 1 | 0 | 0 | 1 | 1 | 34.928 | 55-59 | Obese Class I | 106.666667 | Grade 1 Hypertension |
| **2** | 52 | 1 | 165 | 64.0 | 130 | 70 | 3 | 1 | 0 | 0 | 0 | 1 | 23.508 | 50-54 | Normal Weight | 90.000000 | Optimal |
| **3** | 48 | 2 | 169 | 82.0 | 150 | 100 | 1 | 1 | 0 | 0 | 1 | 1 | 28.710 | 45-49 | Overweight | 116.666667 | Grade 1 Hypertension |
| **4** | 48 | 1 | 156 | 56.0 | 100 | 60 | 1 | 1 | 0 | 0 | 0 | 0 | 23.011 | 45-49 | Normal Weight | 73.333333 | Optimal |

# CATEGORISING DATA

```python
# Define BMI groups with 7 categories
def categorize_bmi(bmi):
    if bmi < 16:
        return 'Severely Underweight'
    elif 16 <= bmi < 17:
        return 'Moderately Underweight'
    elif 17 <= bmi < 18.5:
        return 'Mildly Underweight'
    elif 18.5 <= bmi < 25:
        return 'Normal Weight'
    elif 25 <= bmi < 30:
        return 'Overweight'
    elif 30 <= bmi < 35:
        return 'Obese Class I'
    elif 35 <= bmi < 40:
        return 'Obese Class II'
    else:
        return 'Obese Class III'
```

| | age | gender | heig | active | cardio | BMI | Age Group | BMI Group | MAP | MAP Group |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 50 | 2 | 1 | 1 | 0 | 21.967 | 50-54 | Normal Weight | 90.000000 | Optimal |
| **1** | 55 | 1 | 1 | 1 | 1 | 34.928 | 55-59 | Obese Class I | 106.666667 | Grade 1 Hypertension |
| **2** | 52 | 1 | 1 | 0 | 1 | 23.508 | 50-54 | Normal Weight | 90.000000 | Optimal |
| **3** | 48 | 2 | 1 | 1 | 1 | 28.710 | 45-49 | Overweight | 116.666667 | Grade 1 Hypertension |
| **4** | 48 | 1 | 1 | 0 | 0 | 23.011 | 45-49 | Normal Weight | 73.333333 | Optimal |

# CATEGORISING DATA

```python
# Define function to calculate Mean Arterial Pressure (MAP)
def calculate_map(systolic, diastolic):
    return diastolic + 1/3 * (systolic - diastolic)
```

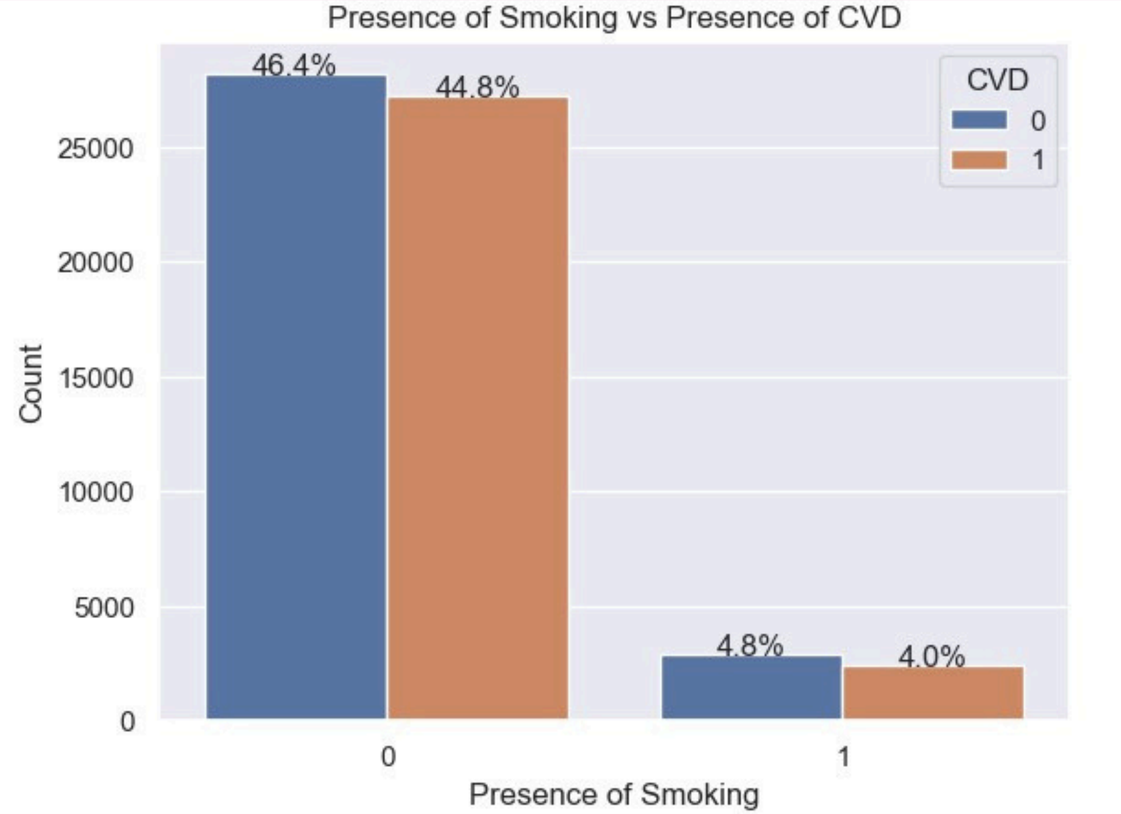
```python
# Define MAP groups with 6 categories
def categorize_map(map_value):
    if map_value < 93.33:
        return 'Optimal'
    elif 93.33 <= map_value < 99.00:
        return 'Normal'
    elif 99.00 <= map_value < 105.67:
        return 'High Normal'
    elif 105.67 <= map_value < 119.00:
        return 'Grade 1 Hypertension'
    elif 119.00 <= map_value < 132.33:
        return 'Grade 2 Hypertension'
    else:
        return 'Grade 3 Hypertension'
```

| | a | | | | | | | BMI | Age Group | BMI Group | MAP | MAP Group |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | 1.967 | 50-54 | Normal Weight | 90.000000 | Optimal |
| 1 | 55 | 1 | | | 0 | 1 | 1 | 34.928 | 55-59 | Obese Class I | 106.666667 | Grade 1 Hypertension |
| 2 | 52 | 1 | | | 0 | 0 | 1 | 23.508 | 50-54 | Normal Weight | 90.000000 | Optimal |
| 3 | 48 | 2 | | | 0 | 1 | 1 | 28.710 | 45-49 | Overweight | 116.666667 | Grade 1 Hypertension |
| 4 | 48 | 1 | | | 0 | 0 | 0 | 23.011 | 45-49 | Normal Weight | 73.333333 | Optimal |

# EXPLORATORY DATA ANALYSIS & VISUALIZATION

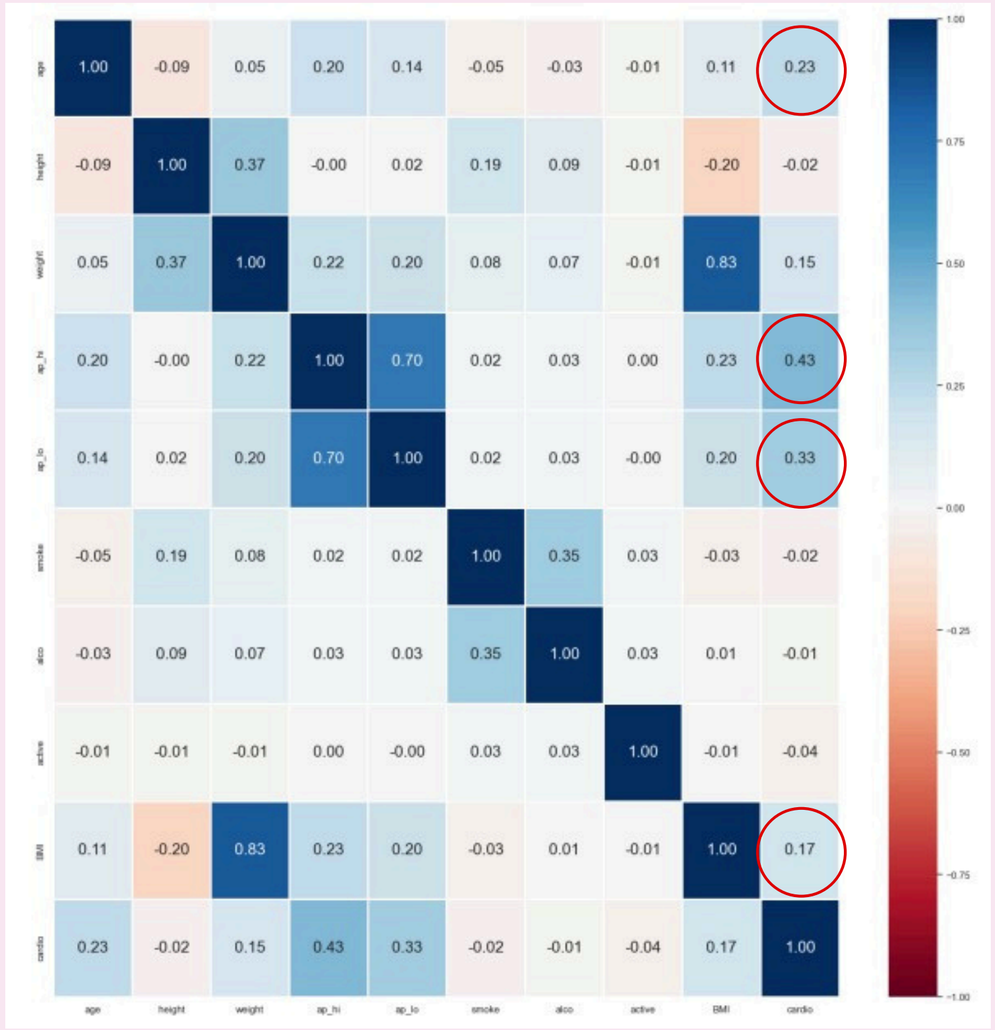
Examining The Effects Of Each Variable
To Cardiovascular Disease

# DATA VISUALIZATION




Percentage difference between people who have CVD, among smokers and non-smokers  are relatively the same

# EXPLORATORY




# TOP CORRELATION WITH CVD

1st - AP_HI (0.43)

2nd - AP_LOW (0.33)

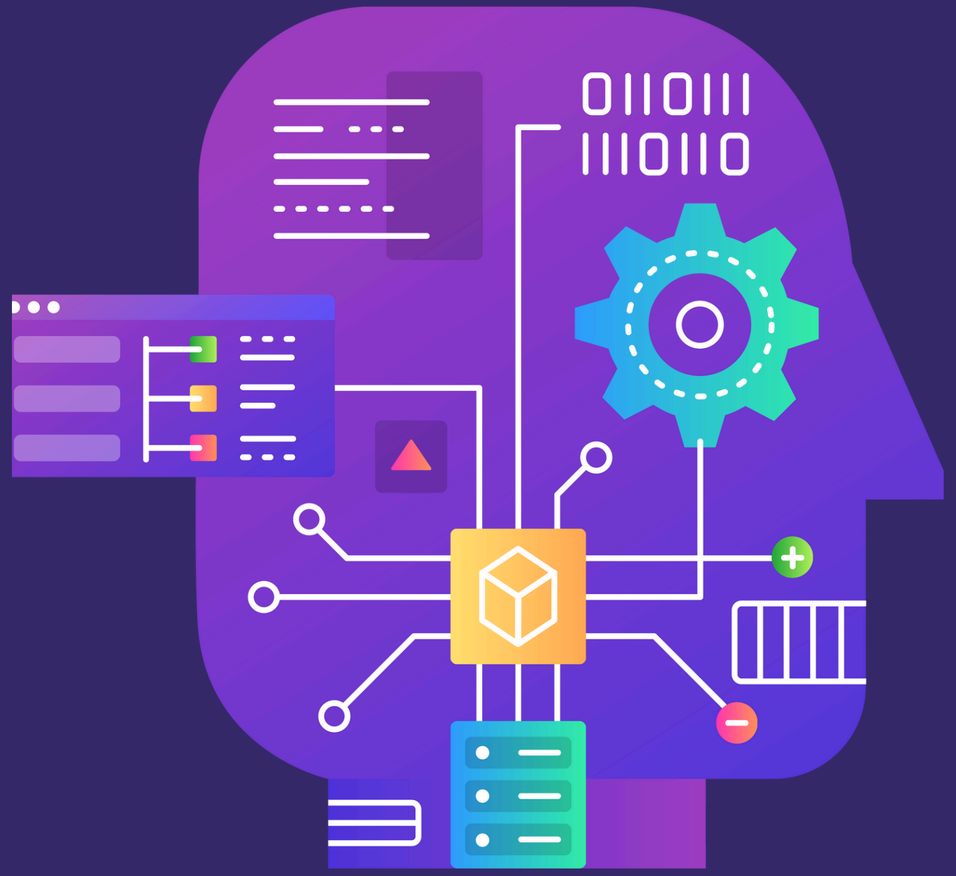
3rd - AGE (0.23)

4th - BMI (0.17)

# EXPLORATORY

# MACHINE LEARNING MODEL

Naive Bayes, K-Nearest Neighbor (KNN) and SVM Classifier



# PREPARATION

## APPLICABLE TO ALL MODELS

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score
```

```
# Randomly choose 10,000 samples from the dataset
cardiodf_sampled = cardiodf.sample(n=10000, random_state=42)

# Define the predictors and response
X = cardiodf_sampled[['BMI', 'ap_hi', 'ap_lo', 'smoke', 'alco', 'active']] # predictors
y = cardiodf_sampled['cardio'] # response

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

# NAIVE BAYES MODEL

## WHAT IS IT?

- Naive Bayes methods are a set of supervised learning algorithm that predicts the probability of a target variable based on various independent predictor variables

## GAUSSIAN NAIVE BAYES

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

```
from sklearn.naive_bayes import GaussianNB
```

```
Gaussian Naive Bayes Model accuracy: 0.52850
```

# K-NEAREST NEIGHBOR (KNN )MODEL

## WHAT IS IT?

- K-Nearest Neighbor (KNN) model is a supervised classification algorithm that assigns class based on the ‘K’ nearest points

```
from sklearn.neighbors import KNeighborsClassifier
```

```
# Training the KNN Model
knn_model = KNeighborsClassifier(n_neighbors=15)  # n_neighbors is k
knn_model.fit(X_train, y_train)
```

```
K-Nearest Neighbor (KNN) Model Accuracy: 0.71850
```

# IMPROVEMENTS

```
from sklearn.model_selection import GridSearchCV
```

```
# Setting up GridSearchCV to find the best n_neighbors with jumps of 5
param_grid = {'n_neighbors': [1] + list(range(5, 501, 5))}
grid_search = GridSearchCV(KNeighborsClassifier(), param_grid, cv=5)
grid_search.fit(X_train, y_train)
```

```
Best n_neighbors: 440
Improved K-Nearest Neighbor (KNN) Model Accuracy: 0.72700
```

- Introducing GridSearchCV to improve the KNN Model
- Simplify process of identifying ‘best’ value for ‘k’
- Improved model has a better accuracy compared to 0.71750 previously

# SVM CLASSIFIER MODEL

## WHAT IS IT?

- Support Vector Machine (SVM) is a powerful and versatile supervised model used for classification and regression, with it’s main goal in finding the hyperplane in an N-dimensional plane that distinctively classifies a data point

```
from sklearn.svm import SVC
```

```
# Define the parameter grid
param_grid = {'C': [0.1, 1, 10, 100, 1000]}

# Create and fit the GridSearchCV
grid_search = GridSearchCV(SVC(), param_grid, cv=3)
grid_search.fit(X_train, y_train)
```

```
SVM Classifier Model Accuracy: 0.72200
```

# CONCLUSION

# CONCLUSION

| | Accuracy |
|---|---|
| Gaussian Naive Bayes | 0.52850 |
| Improved K-Nearest Neighbors (KNN) | 0.72700 |
| SVM Classifier | 0.72250 |

- K-Nearest Neighbor (KNN) model has the most accurate model amongst the three models
- KNN model will be utilized for the prediction of Cardio Vascular Disease

# CONCLUSION

## Final Prediction

```
Height (cm): 180
Weight (kg): 70
Systolic blood pressure (ap_hi): 130
Diastolic blood pressure (ap_lo): 70
Do you smoke? Type 1 for Yes and 0 for No: 1
Do you consume alcohol? Type 1 for Yes and 0 for No: 0
Are you physically active? Type 1 for Yes and 0 for No: 1
You are predicted to be at low risk of cardiovascular disease.
```

# THANK YOU!

# CONTRIBUTIONS

| | Codes | Slides |
|---|---|---|
| JONAS | DATA CLEANING | 4 to 14 |
| SHUKERY | Exploratory data analysis & visualization | 13 to 17, 24 to 26 |
| CHUAN ZHEN | MACHINE LEARNING MODEL | 18 TO 23 |

# REFERENCES


- BMI categories: https://www.ncbi.nlm.nih.gov/books/NBK551660/figure/article-35266.image.f1/
- MAP categories: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10046034/#:~:text=An%20optimal%20MAP%20value%20is,MAP%20values%20are%20%E2%89%A5132.34.
- Naive Bayes: https://scikit-learn.org/stable/modules/naive_bayes.html
- K-Nearest Neighbor: https://scikit-learn.org/stable/modules/neighbors.html , https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html , https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
- SVM Classifier: https://scikit-learn.org/stable/modules/svm.html , https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html , https://scikit-learn.org/stable/modules/grid_search.html